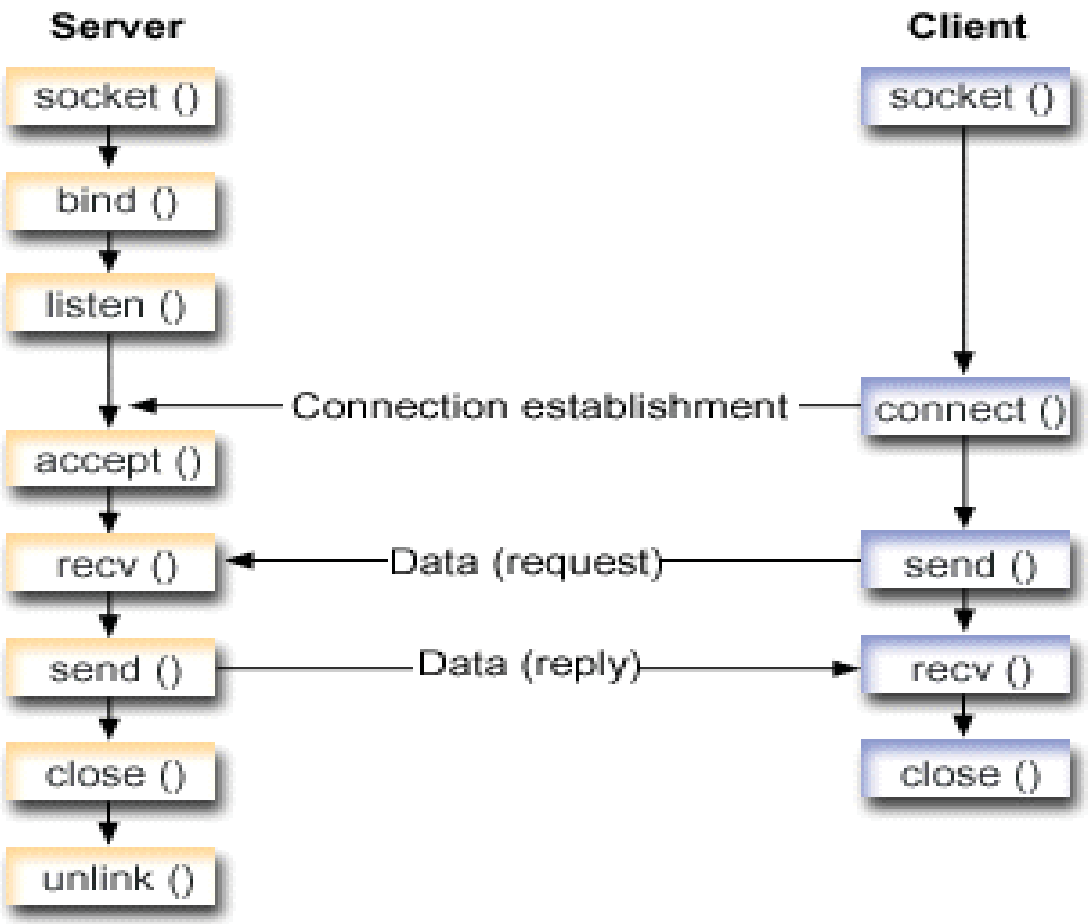
- client/server con socket connection oriented -



- Esercizi: client/server con Socket locale - (STREAM SOCKET)

- client/server: con server concorrente multi processo per la gestione di più client-

Esercizio n° 4 - Socket locali-

- Scrivere due programmi C, **server.c** e **client.c**. che comunicano tramite socket. Il **server.c** crea una socket locale, ed ogni volta che instaura una connessione con un client, mediate **fork** crea un nuovo processo (server concorrente). Tale processo, dovrà leggere sul socket il messaggio scritto dal client e scrivere il messaggio che il server manda al client. Il **client.c** dovrà connettersi al socket locale su cui scriverà il messaggio da mandare al server.

  (lanciare l' esecuzione dei due programmi su due shell distinti della stessa macchina)

  Esecuzione

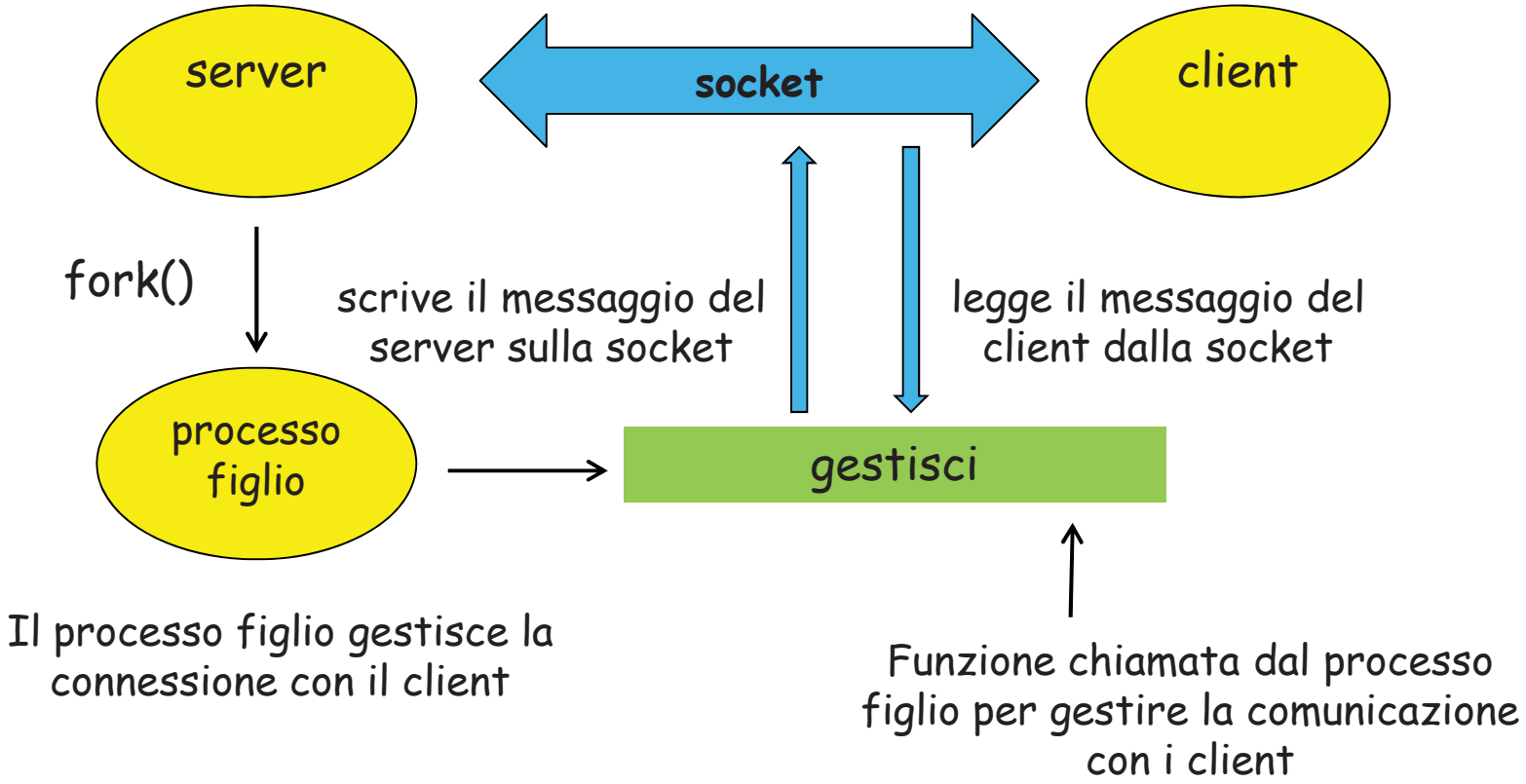
  $ (shell1) ./server.out MESSAGGIO DA CLIENT: <messaggio>

  $ (shell2) ./client.out <messaggio>

Suggerimento:
Si utilizzi **fgets** per ricevere il messaggio da riga di comando

- Comunicazione client/server con socket locale -

Schema Server concorrente multi-processo per gestire più client



- Parametri per creare un socket locale connection oriented -

```
#include <sys/socket.h>
int socket(int famiglia, int tipo, int protocollo);

famiglia = PF_LOCAL oppure PF_UNIX
tipo = SOCK_STREAM  ( paradigma connection oriented )
protocollo = 0 ( protocollo di default )
int socket( PF_LOCAL, SOCK_STREAM, 0 );
```

- Indirizzamento:

```
#include <sys/un.h>
struct sockaddr_un serverAddress;

serverAddress.sun_family = AF_LOCAL;
strcpy(serverAddress.sun_path, "/tmp/mio_socket");

bind(fd,(struct sockaddr *)serverAddress,
     sizeof(serverAddress));
```

Sol. Eser. n° 4 server.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/types.h>
#include <unistd.h>
int gestisci(int connection_fd)
{
    int nbytes;
    char buffer[256];
    //legge da socket il mess. del client
    nbytes = read(connection_fd, buffer, 256);
    buffer[nbytes] = 0;
    printf("MESSAGGIO DA CLIENT: %s\n", buffer);
    nbytes = sprintf(buffer, "Saluti dal server");
    //Scrive su socket il mess. del server
    write(connection_fd, buffer, nbytes);
    close(connection_fd);
     return 0;
}
```

Funzione chiamata dal processo figlio per comunicare con il client tramite socket

sockfd

Sol. Eser. n° 4 server.c

```
int main(void)   //server
{//struttura degli indirizzi di tipo locale
 struct sockaddr_un serverAddress;
 //Puntatore alla struttura degli indirizzi
 struct sockaddr* serverSockAddrPtr;
 int socket_fd, conn_fd, serverAddressLen;
 pid_t child;

 socket_fd = socket(PF_LOCAL, SOCK_STREAM, 0);//crea socket
 if(socket_fd < 0)
 {printf("socket() failed\n");
  return 1;}
 unlink("demo_socket"); /*Rimuove la socket se già esiste */
 //valorizzo i campi della struttura sockaddr_un
 serverAddress.sun_family = AF_LOCAL; //setta il tipo di dominio
 sprintf(serverAddress.sun_path, "demo_socket");

//Parametri da passare all funzione bind
 serverSockAddrPtr = (struct sockaddr*) &serverAddress;
 serverAddressLen = sizeof (serverAddress);
```

Orientato alla connessione

Sol. Eser. n° 4 server.c

```
if(bind(socket_fd, serverSockAddrPtr, serverAddressLen) != 0)
 {printf("bind() failed\n");
  return 1;}
 if(listen(socket_fd, 5) != 0)//si mette in ascolto sulla coda
 {printf("listen() failed\n");
  return 1;}

  //crea la connessione con i client
  while((conn_fd = accept(socket_fd,NULL,NULL)) > -1)
 {
    printf("server: new connection from %d \n",conn_fd);
    child = fork();//server concorrente crea il figlio
    if(child == 0){
    return (gestisci(conn_fd)); //comunica con client
    }
    close(conn_fd);
 }//end while
 close(socket_fd);
 unlink("demo_socket"); /*Rimuove la socket*/
 return 0;
}//end server
```

Sol. Eser. n° 4 client.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
int main(void) //client
{struct sockaddr_un serverAddress;
 //Puntatore alla struttura degli indirizzi
 struct sockaddr* serverSockAddrPtr;
 int socket_fd, nbytes, serverAddressLen;
 char buffer[256];
 socket_fd = socket(PF_LOCAL, SOCK_STREAM, 0);//crea socket
 if(socket_fd < 0)
 {printf("socket() failed\n");
  return 1;}
 //valorizza i campi della struttura indirizzi
 serverAddress.sun_family = AF_LOCAL;
 sprintf(serverAddress.sun_path, "demo_socket");
//Parametri da passare all funzione connect
 serverSockAddrPtr = (struct sockaddr*) &serverAddress;
 serverAddressLen = sizeof (serverAddress);
```

Orientato alla connessione

Sol. Eser. n° 4 client.c

```
//si propone di connettersi al socket del server
if(connect(socket_fd, serverSockAddrPtr, serverAddressLen) != 0)
 {
  printf("connect() failed\n");
  return 1;
 }
printf("(CLIENT) Scrivere un messaggio: ");
fgets(buffer,255,stdin); //immesso da tastiera
 //scrive su socket il mess. da mandare al server
 write(socket_fd, buffer, strlen(buffer));
 //legge da socket il mess. del server
 nbytes = read(socket_fd, buffer, 256);
 buffer[nbytes] = '\0';

 printf("MESSAGGIO DA SERVER: %s\n", buffer);

 close(socket_fd);

 return 0;
}//end client
```

- client/server: con server concorrente multi thread per la gestione di più client -

- Struttura di un server concorrente multiThread -

```
socket(...);
bind(...);
listen(...);

while (1) {//loop principale

connect_sd = accept(listen_sd, NULL,NULL)
printf("server: new connection from %d \n",connect_sd);
pthread_create(&tid,NULL,gestisci,(void *)connect_sd);

}

void *gestisci(void *arg) {
/* *** fa quello che deve fare *** */
/* il thread termina se stesso */
pthread_exit(0);
}
```

Esercizio n° 5 - Socket locali-

- Scrivere due programmi C, **serverMultiThread.c** e **client.c**. Il programma **serverMultiThread.c**, crea una socket locale e rimane in attesa di ricevere una connessione. Quando riceve una connessione, crea un **thread** con chiamata ad una funzione che gestisce la connessione, mentre il programma server rimane in ascolto per eventuali altre richieste di connessione. Il programma client, si connette alla socket locale ed invia il messaggio passato a riga di comando (usare fgets) al server. Il server tramite il thread legge il messaggio e lo stampa a video

  (Lanciare i due programmi in due shell distinte, e compilare il server con -lpthread)

Esecuzione server:

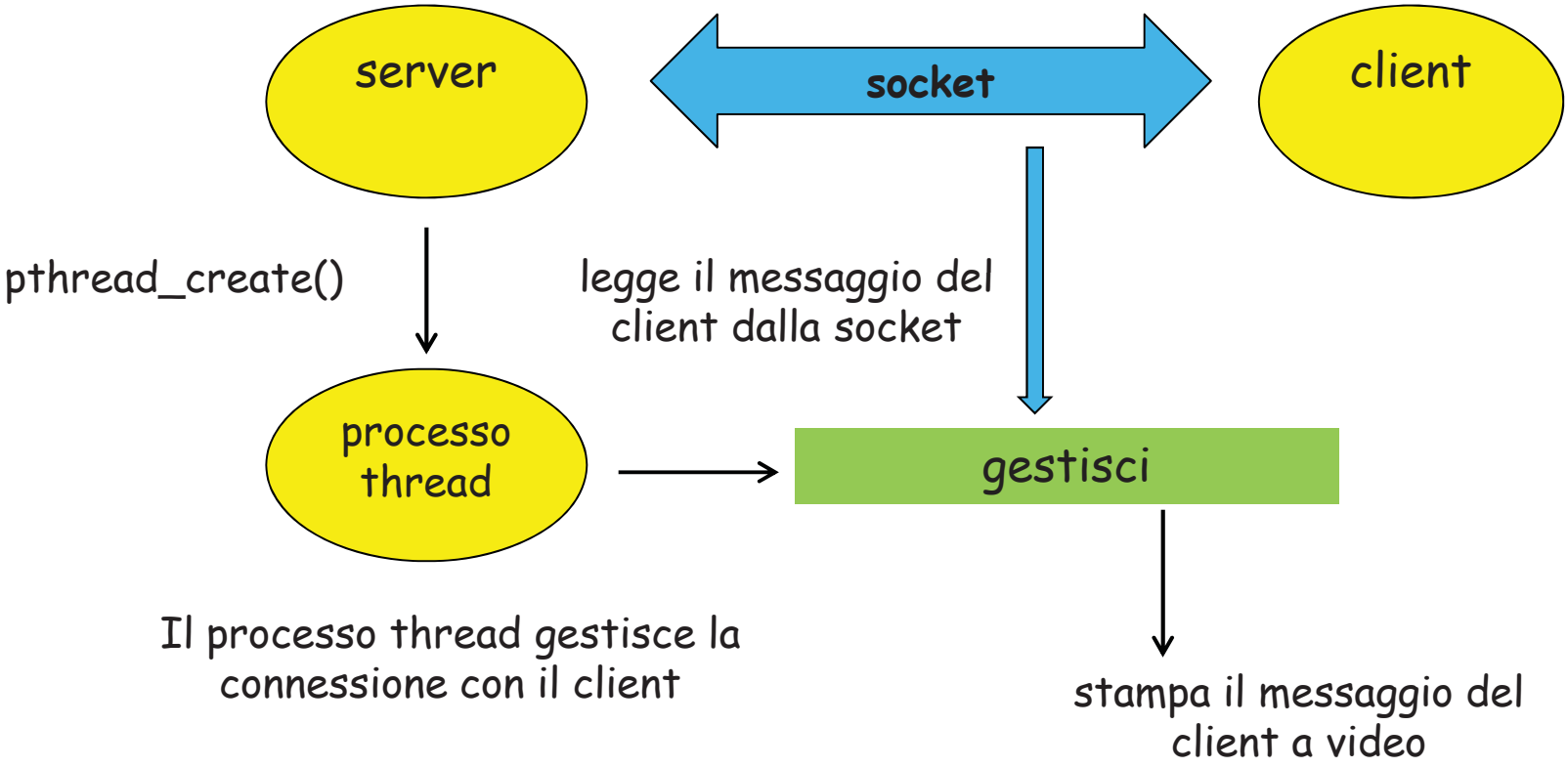
shell1 $ ./serverMultiThread.out

Esecuzione client:

shell2 $ ./client.out "Messaggio"

- Comunicazione client/server con socket locale -

Schema Server concorrente multi-thread per gestire più client



Sol. Eser. n° 5 serverMultiThread.c

```
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <pthread.h>

#define SOCKNAME "mysocket"
#define N 256

void *gestisci(void *arg) {
printf("gestisci(): creato il thread\n");
char buf[N];
//Legge da socket il messaggio scritto dal client
while (read((int *) arg, buf, N) != 0)
{
   printf("Server %d : %s\n", getpid(), buf);
}
close((int *) arg);
pthread_exit (0);
}
```

Funzione chiamata dal thread per comunicare con il client tramite socket

Sol. Eser. n° 5 serverMultiThread.c

```
int main(void)
{unlink(SOCKNAME);
int s_fd, c_fd;
pthread_t tid;
struct sockaddr_un sa;
sprintf(sa.sun_path, SOCKNAME);
sa.sun_family = AF_LOCAL;

s_fd = socket(AF_LOCAL, SOCK_STREAM, 0);
bind(s_fd, (struct sockaddr *) &sa, sizeof(sa));
listen(s_fd, 5);
while (1) { //loop infinito

    c_fd = accept(s_fd, NULL, NULL);
    printf("server: new connection from %d \n",c_fd);
    pthread_create(&tid, NULL, gestisci, (void *) c_fd)
    pthread_detach(tid);

 }
return 0;
}//end server
```

Passaggio dell' id socket alla funzione gestisci

Sol. Eser. n° 5 client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define SOCKNAME "mysocket”
#define N 256

int main(void)
{int c_fd;
struct sockaddr_un sa;
char buffer[N];
sprintf(sa.sun_path, SOCKNAME);
sa.sun_family = AF_UNIX;

c_fd = socket(AF_UNIX, SOCK_STREAM, 0);
```

Sol. Eser. n° 5 client.c

```
if(connect(c_fd, (struct sockaddr *) &sa, sizeof(sa)) < 0)
{
    printf("connect() failed\n");
     return 1;
}

printf("(CLIENT) Scrivere un messaggio: ");

fgets(buffer,N,stdin); //Inserimento del messaggio

write(c_fd, buffer, strlen(buffer)); //scrittura su socket

printf("Client %d ha inviato il messaggio\n", getpid());

close(c_fd);

return 0;

}//end client
```

- client/server: con server interattivo -

Esercizio n° 6 - Socket locali-

- Scrivere due programmi C, **socket-server.c** e **socket-client.c**. Il programma **socket-server.c**, crea una socket locale e rimane in attesa di ricevere una connessione. Quando riceve una connessione, legge il messaggio proveniente da essa, lo stampa e chiude la connessione. Se il messaggio ricevuto è "**quit**", il programma server, rimuove la socket e termina l'esecuzione. Il server prende il path della socket da linea di comando. Il programma client, si connette alla socket locale ed invia il messaggio. Il nome del path della socket e del messaggio da inviare sono specificati a linea di comando. (Lanciare i due programmi in due shell distinte)

Esecuzione server:

```
$ ./socket-server  demo_socket
```

Esecuzione client:

```
shell1 $ ./socket-client   demo_socket  "Messaggio"
shell2 $ ./socket-client   demo_socket   "quit"
```

Sol. Eser. n° 6 socket-server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
int server (int client_socket)
{   while (1)
    {  int length;
       char* text;
       if (read (client_socket, &length, sizeof (length)) == 0)
            return 0;
       text = (char*) malloc (length); /*Alloca memoria per il buffer*/
       read (client_socket, text, length);/*legge il messaggio e lo mette in text*/
       printf ("%s\n", text);
      free (text);  /*Pulisce il buffer */
         if (!strcmp (text, "quit"))  /*Se text è quit termina*/
                return 1;
    }
}
```

chiamata dal server per leggere il mess. del client nel socket

Legge la lunghezza del messaggio dalla socket e la mette in length

Sol. Eser. n° 6 socket-server.c

```
int main (int argc, char* const argv[])
{
    const char* const socket_name = argv[1]; //nome socket
    struct sockaddr_un serverUNIXAddress;  /*Server address*/
    struct sockaddr* serverSockAddrPtr; /*Ptr to server address*/
    int socket_fd, serverLen, clientLen;
    int client_sent_quit_message;

    socket_fd = socket (PF_LOCAL, SOCK_STREAM, 0);//creazione socket

    //Valorizzo la struttura degli indirizzi
    serverUNIXAddress.sun_family = AF_LOCAL;
    strcpy (serverUNIXAddress.sun_path, socket_name);

    //Parametri da passare all funzione bind
    serverSockAddrPtr = (struct sockaddr*) &serverUNIXAddress;
    serverLen = sizeof (serverUNIXAddress);

    unlink (socket_name);
    bind (socket_fd, serverSockAddrPtr, serverLen);
    listen (socket_fd, 5); /* coda di attesa */
```

Sol. Eser. n° 6 socket-server.c

```
do {
   int client_socket_fd;

   client_socket_fd = accept (socket_fd, NULL, NULL);

   client_sent_quit_message = server(client_socket_fd);
   close (client_socket_fd);

    }  while (!client_sent_quit_message);//cicla se non quit

close (socket_fd);
unlink (socket_name);  /*Rimuove la socket*/

return 0;

} //END SERVER
```

Sol. Eser. n° 6 socket-client.c

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

void write_text (int socket_fd, const char* text)
{
   int length = strlen (text) + 1;
  /*Write the number of bytes in the string,
   including NUL-termination.*/
   write (socket_fd, &length, sizeof (length));
   write (socket_fd, text, length);
}
```

chiamata dal client per scrivere il mess. nel socket

Sol. Eser. n° 6 socket-client.c

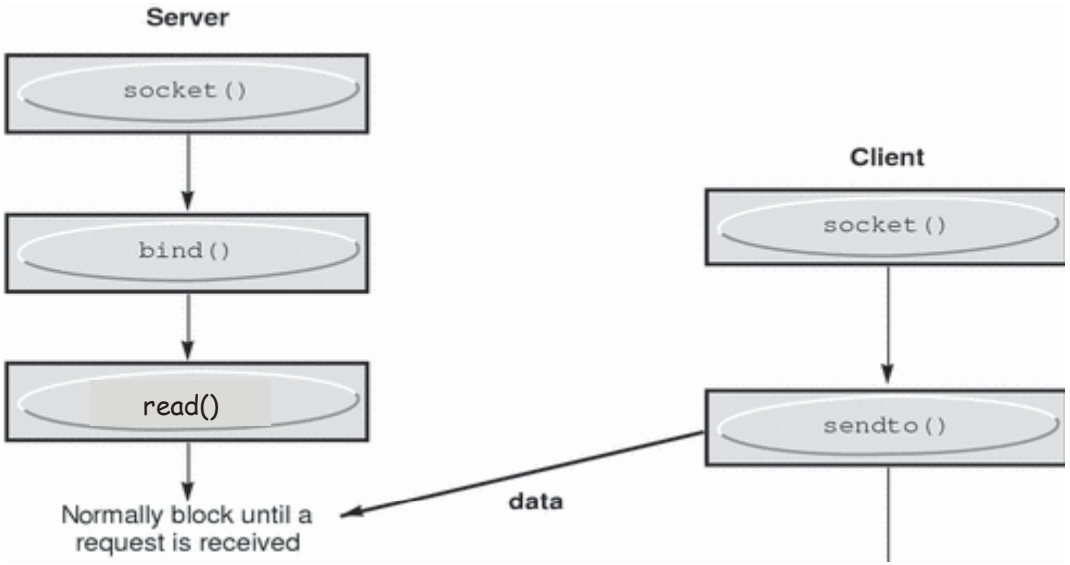
```
int main (int argc, char* const argv[])
{
   const char* const socket_name = argv[1];//nome socket
   const char* const message = argv[2]; //messaggio
   int serverLen, socket_fd;
   struct sockaddr_un   serverUNIXAddress;
   struct sockaddr*   serverSockAddrPtr;
   serverSockAddrPtr = (struct sockaddr*) &serverUNIXAddress;
   serverLen = sizeof (serverUNIXAddress);

   socket_fd = socket (PF_LOCAL, SOCK_STREAM, 0); //crea socket

   serverUNIXAddress.sun_family = AF_LOCAL;
   strcpy (serverUNIXAddress.sun_path, socket_name);
   connect (socket_fd, serverSockAddrPtr, serverLen);
   write_text (socket_fd, message);//scrittura mess. su socket
   close (socket_fd);
   return 0;
} //end client
```

- client/server con Socket locale - connectionless (DATAGRAM SOCKET)

- client/server con DATAGRAM SOCKET -



Funzione: sendto

- Funzione **sendto()**:

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto(int sockfd, const void *buf, size_t len, int flags,
           const struct sockaddr *to, socklen_t tolen);
```

- Manda un messaggio su datagram socket (senza connessione)
- **sockfd**: è il descrittore della socket su cui mandare il messaggio
- **buf**: è il puntatore al messaggio da mandare
- **len**: è la dimensione del messaggio
- **flags**: consente di specificare altre info sul messaggio da mandare; per messaggi normali viene posto a zero
- **to**: puntatore alla struttura contenete l'indirizzo del destinatario del messaggio
- **tolen**: dimensione struttura
- Ritorna
  - il numero di byte mandati se ok
  - -1 altrimenti

- Esercizi: client/server con Socket locale - (DATAGRAM SOCKET)

Esercizio n° 7 - Socket locali -

- Scrivere due programmi C, **server.c** e **client.c**. che comunicano tramite DATAGRAM socket. Il **server.c** crea una socket locale, e si mette in attesa di leggere un messaggio sulla socket. Il **client.c** scrivere un messaggio su socket.

  Il nome della socket viene passato al client da riga di comando (lanciare l' esecuzione dei due programmi su due shell distinti della stessa macchina)

  Esecuzione

  $ ./server.out (shell1) MESSAGGIO DA CLIENT: Saluti da client

  $ ./client.out socket (shell2)

Sol. Eser. n° 7 server.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdlib.h>
#include <stdio.h>
#define NAME "socket" //nome della socket
main()
{
int sock, length;
struct sockaddr_un name;
char buf[1024]; /* Create socket from which to read. */
sock = socket(PF_LOCAL, SOCK_DGRAM, 0);
if (sock < 0) {
perror("opening datagram socket");
exit(1);    }
/* Create name. */
name.sun_family = AF_LOCAL;
strcpy(name.sun_path, NAME);
```

Sol. Eser. n° 7 server.c

```
/* Bind the UNIX domain address to the created socket */
if (bind(sock, (struct sockaddr *) &name,
         sizeof(struct sockaddr_un)))
{
    perror("binding name to datagram socket");
    exit(1);    }

printf("socket -->%s\n", NAME);

/* Read from the socket */
if (read(sock, buf, 1024) < 0)
    perror("receiving datagram packet");

 printf("-->%s\n", buf);//STAMPA DEL MESSAGGIO LETTO

close(sock);

unlink(NAME);

}//END SERVER
```

Soluzione Esercizio N° 7 client.c 1/1

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#define DATA "socket locale connectionless”
main(argc, argv)
int argc;
char *argv[];
{    int sock;
struct sockaddr_un name;
sock = socket(PF_LOCAL, SOCK_DGRAM, 0); //Crea socket dove scrivere
if (sock < 0) {
perror("opening datagram socket");
exit(1);    }
/* Costruzione indirizzo */
name.sun_family = AF_LOCAL;
strcpy(name.sun_path, argv[1]);
/* manda message. */
if (sendto(sock, &DATA, sizeof(DATA), 0, (struct sockaddr *)&name,
        sizeof(name)) < 0)
{ perror("sending datagram message");}
close(sock);
}//END CLIENT
```

- Fine Esercitazione -